**Fundamentos VueJS**

**Instructor**: Abner Saavedra

**Fecha**: sept. de 2024

**Email**: [ingenieroabnersaavedra@gmail.com](mailto:ingenieroabnersaavedra@gmail.com)

**GracoSoft** Centro Empresarial Plaza Madrid, piso 9, oficinas 9-7 a la 9-10

# Agenda

## Rutas y controladores

## Definición de Rutas

## Parámetros de Ruta

## Controladores en Express.js

## Middleware de Rutas

## Rutas Anidadas y Modularización

## Ejemplo Completo

**¿Qué es Vue?**

Vue (pronunciado /vjuː/, como view) es un marco de JavaScript para crear interfaces de usuario. Se basa en HTML, CSS y JavaScript estándar y proporciona un modelo de programación declarativo basado en componentes que lo ayuda a desarrollar de manera eficiente interfaces de usuario de cualquier complejidad.

* Vue es un marco (Framework) de JavaScript para construir interfaces de usuario.
* Se basa en HTML, CSS y JavaScript estándar, y proporciona un modelo de programación declarativo y basado en componentes que lo ayuda a desarrollar interfaces de usuario de manera eficiente, ya sea simple o compleja.

He aquí un ejemplo mínimo:

**js**

**import** { createApp, ref } from 'vue'

createApp({

setup() {

**return** {

count: ref(0)

}

}

}).mount('#app')

**plantilla**

<div id="app">

<button @click="count++">

Count is: {{ count }}

</button>

</div>

Resultado

El conteo es: 0

El ejemplo anterior demuestra las dos características principales de Vue:

* **Representación declarativa** : Vue extiende el HTML estándar con una sintaxis de plantilla que nos permite describir de forma declarativa la salida HTML en función del estado de JavaScript.
* **Reactividad** : Vue rastrea automáticamente los cambios de estado de JavaScript y actualiza eficientemente el DOM cuando ocurren cambios.

Es posible que ya tengas preguntas, no te preocupes. Trataremos cada pequeño detalle más adelante.

**El marco progresista**

Vue es un marco y un ecosistema que cubre la mayoría de las características comunes necesarias en el desarrollo de frontend. Pero la web es extremadamente diversa: las cosas que creamos en la web pueden variar drásticamente en forma y escala. Con eso en mente, Vue está diseñado para ser flexible y adoptarse de manera incremental. Según su caso de uso, Vue se puede utilizar de diferentes maneras:

* Mejorar HTML estático sin un paso de compilación
* Incrustar como componentes web en cualquier página
* Solicitud de una sola página (SPA)
* Representación fullstack/del lado del servidor (SSR)
* Jamstack / Generación de sitios estáticos (SSG)
* Orientado a computadoras de escritorio, dispositivos móviles, WebGL e incluso terminales

Si estos conceptos te intimidan, no te preocupes. Este curso solo requerirá conocimientos básicos de HTML y JavaScript, y deberías poder seguirlos sin ser un experto en ninguno de ellos.

**Componentes de un solo archivo**[​](https://vuejs.org/guide/introduction.html#single-file-components)

En la mayoría de los proyectos de Vue habilitados para herramientas de compilación, creamos componentes de Vue utilizando un formato de archivo similar a HTML llamado **componente de archivo único** (también conocido como \*.vuearchivos, abreviado como **SFC** ). Un SFC de Vue, como sugiere el nombre, encapsula la lógica del componente (JavaScript), la plantilla (HTML) y los estilos (CSS) en un solo archivo. Aquí está el ejemplo anterior, escrito en formato SFC:

<script setup>

**import** { ref } from 'vue'

**const** count = ref(0)

</script>

<template>

<button @click="count++">Count is: {{ count }}</button>

</template>

<style scoped>

button {

font-weight: **bold**;

}

</style>

SFC es una característica definitoria de Vue y es la forma recomendada de crear componentes Vue si su caso de uso justifica una configuración de compilación. Puede obtener más información sobre el [cómo y el por qué de SFC](https://vuejs.org/guide/scaling-up/sfc.html) en su sección dedicada, pero por ahora, solo sepa que Vue se encargará de toda la configuración de las herramientas de compilación por usted.

**Estilos API**[**​**](https://vuejs.org/guide/introduction.html#api-styles)

Los componentes de Vue se pueden crear en dos estilos de API diferentes: API de opciones y API de composición.

**API de Opciones** [**​**](https://vuejs.org/guide/introduction.html#options-api)

Con la API de opciones, definimos la lógica de un componente mediante un objeto de opciones como data, methodsy mounted. Las propiedades definidas por las opciones se exponen en thisfunciones internas, que apuntan a la instancia del componente:

<script>

**export** **default** {

// Properties returned from data() become reactive state

// and will be exposed on `this`.

data() {

**return** {

count: 0

}

},

// Methods are functions that mutate state and trigger updates.

// They can be bound as event handlers in templates.

methods: {

increment() {

**this**.count++

}

},

// Lifecycle hooks are called at different stages

// of a component's lifecycle.

// This function will be called when the component is mounted.

mounted() {

console.**log**(`The initial count is ${**this**.count}.`)

}

}

</script>

<template>

<button @click="increment">Count is: {{ count }}</button>

</template>

**API de composición**[​](https://vuejs.org/guide/introduction.html#composition-api)

Con Composition API, definimos la lógica de un componente mediante funciones API importadas. En los SFC, Composition API se usa normalmente con [<script setup>](https://vuejs.org/api/sfc-script-setup.html). El setupatributo es una sugerencia que hace que Vue realice transformaciones en tiempo de compilación que nos permiten usar Composition API con menos código repetitivo. Por ejemplo, las importaciones y las variables/funciones de nivel superior declaradas en <script setup>se pueden usar directamente en la plantilla.

Aquí está el mismo componente, con exactamente la misma plantilla, pero usando Composition API y <script setup>en su lugar:

<script setup>

**import** { ref, onMounted } from 'vue'

// reactive state

**const** count = ref(0)

// functions that mutate state and trigger updates

**function** increment() {

count.value++

}

// lifecycle hooks

onMounted(() => {

console.**log**(`The initial count is ${count.value}.`)

})

</script>

<template>

<button @click="increment">Count is: {{ count }}</button>

</template>

## Parámetros de Ruta

Los parámetros de ruta permiten a tus rutas capturar valores específicos desde la URL.

**Ejemplo de uso de parámetros de ruta:**

// Definir una ruta con un parámetro de ruta

app.get('/user/:id', (req, res) => {

res.send(`Usuario ID: ${req.params.id}`);

});

Cuando navegas a /user/123, la respuesta será "Usuario ID: 123".

## Controladores en Express.js

Para mantener nuestro código limpio y organizado, es una buena práctica separar la lógica de nuestras rutas en controladores. Los controladores son simplemente módulos que exportan funciones manejar la lógica de las rutas.

**Estructura del proyecto:**

mi-proyecto-express

|-- app.js

|-- routes

|-- index.js

|-- controllers

|-- userController.js

1. **Crear el controlador:**

*// controllers/userController.js*

exports.getUser = (req, res) => {

res.send(`Usuario ID: ${req.params.id}`);

};

1. **Definir la ruta y utilizar el controlador:**

*// routes/index.js*

const express = require('express');

const router = express.Router();

const userController = require('../controllers/userController');

router.get('/user/:id', userController.getUser);

module.exports = router;

1. **Usar las rutas en la aplicación principal:**

*// app.js*

const express = require('express');

const app = express();

const indexRouter = require('./routes/index');

app.use('/', indexRouter);

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Servidor ejecutándose en http://localhost:${PORT}`);

});

## Middleware de Rutas

Puedes usar middlewares específicos para determinadas rutas para realizar tareas previas a la ejecución del controlador.

**Ejemplo de middleware de ruta:**

*// Middleware específico para la ruta /admin*

const adminMiddleware = (req, res, next) => {

console.log('Se ha accedido a la ruta /admin');

next();

};

*// Usar middleware en la ruta*

app.get('/admin', adminMiddleware, (req, res) => {

res.send('

# 

**Rutas Anidadas y Modularización**

Si tienes una aplicación grande, es una buena práctica modularizar tus rutas en diferentes archivos y carpetas.

**Estructura del proyecto modularizado:**

mi-proyecto-express

|-- app.js

|-- routes

| |-- index.js

| |-- userRoutes.js

|-- controllers

| |-- userController.js

1. **Definir rutas anidadas:**

*// routes/userRoutes.js*

const express = require('express');

const router = express.Router();

const userController = require('../controllers/userController');

router.get('/:id', userController.getUser);

module.exports = router;

1. **Añadir rutas anidadas a la aplicación principal:**

*// routes/index.js*

const express = require('express');

const router = express.Router();

const userRoutes = require('./userRoutes');

router.use('/user', userRoutes);

module.exports = router;

1. **Configurar la aplicación para usar el conjunto de rutas modularizadas:**

*// app.js*

const express = require('express');

const app = express();

const indexRouter = require('./routes/index');

app.use('/', indexRouter);

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Servidor ejecutándose en http://localhost:${PORT}`);

});

**Ejemplo Completo**

A continuación, un ejemplo completo que combina varios de estos conceptos:

**Estructura del proyecto:**

mi-proyecto-express

|-- app.js

|-- routes

| |-- index.js

| |-- userRoutes.js

|-- controllers

| |-- userController.js

1. **app.js:**

const express = require('express');

const app = express();

const indexRouter = require('./routes/index');

app.use(express.json());

app.use('/', indexRouter);

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Servidor ejecutándose en http://localhost:${PORT}`);

});

1. **controllers/userController.js:**

exports.getUser = (req, res) => {

res.send(`Usuario ID: ${req.params.id}`);

};

exports.createUser = (req, res) => {

const { username, email } = req.body;

res.send(`Usuario creado: ${username}, Email: ${email}`);

};

1. **routes/userRoutes.js:**

const express = require('express');

const router = express.Router();

const userController = require('../controllers/userController');

router.get('/:id', userController.getUser);

router.post('/', userController.createUser);

module.exports = router;

1. **routes/index.js:**

const express = require('express');

const router = express.Router();

const userRoutes = require('./userRoutes');

router.use('/user', userRoutes);

router.get('/', (req, res) => {

res.send('Página principal');

});

module.exports = router;

Con este conocimiento sobre rutas y controladores en Express.js, puedes organizar y gestionar la lógica de tu aplicación de manera eficiente, modular y escalable.

# Ejercicio

Construir una API utilizando Node.js y Express para gestionar productos. La API debe permitir realizar las operaciones básicas de creación, lectura, actualización y eliminación (CRUD) de productos

## Requisitos:

1. **Creación de Producto:**
   * Implementa una ruta para agregar un nuevo producto a la base de datos.
   * El cuerpo de la solicitud (body) debe contener la información necesaria del producto, como nombre, precio y cantidad.

## Consulta de Producto:

* + Implementa una ruta para consultar la información de un producto específico por su identificador único.
  + La información del producto debe incluir al menos el nombre, precio y cantidad.

## Modificación de Producto:

* + Crea una ruta para actualizar la información de un producto existente.
  + El cuerpo de la solicitud debe contener los campos que se desean actualizar.

## Eliminación de Producto:

* + Implementa una ruta para eliminar un producto según su identificador único.

## Filtrado de Información:

* + Proporciona la capacidad de filtrar la información de los productos según ciertos criterios (por ejemplo, por precio o cantidad).
  + Los criterios de filtrado deben ser parámetros de la URL.

## Notas Adicionales:

* Utiliza Express para crear el servidor web.
* Utiliza el módulo Express Router para crear las rutas de acceso e implementa controladores para modularizar el código.
* Puedes almacenar la información de los productos en una base de datos simple, como un archivo JSON o en memoria.

# Bibliografía

# <https://bluuweb.github.io/vue-udemy/30-01-fundamentos/#objetivos>

# <https://vuejs.org/guide/introduction.html#what-is-vue>

**GRACOSOFT ES EXCELENCIA EDUCATIVA**